



# Alabama Medicaid

**Health Insurance Portability and Accountability Act (HIPAA)**



## Alabama HIPAA Remediation

---

## Vendor Interface Specification

**EDS**  
**301 Technacenter Drive**  
**Montgomery, AL 36117, USA**

## TABLE OF CONTENTS

1	Version History.....	3
2	Introduction .....	4
2.1	Background.....	4
2.2	HIPAA Approved Transaction Processing.....	4
2.3	Network Security.....	5
2.4	Web Interface .....	5
2.5	Client Software for Web Communications .....	6
2.6	Client Software for Frame Relay Submissions.....	6
2.7	Use of the Alabama Medicaid RAS .....	6
3	Web Interface Specification .....	6
3.1	Connecting to the server.....	6
3.2	HTTPS Messages .....	7
3.3	File Upload Message – X12 Transactions only.....	7
3.4	File Upload Message – X12 and Non-X12 Transactions.....	11
3.5	Directory List Message.....	14
3.6	Download a File Message .....	21
3.7	General Response Message .....	23
4	Frame Relay Interface Specification.....	26
	Appendix A – Sample Upload a File Program .....	27
	Appendix B – Directory Listing sample program.....	37
	Appendix C – Download a File sample program .....	45
	Appendix D – SSL Tunnel through proxy server utility sample program.....	53
	Appendix E – Setting up a RAS Connection.....	59

## 1 Version History

Date	Version	Author	Comments
06/23/2003	1.0	Linda Bragg	Create Initial Document.
09/11/2003	1.1	Linda Bragg	<p>Section 2.2 contains all of the transaction batch types that can be uploaded and downloaded via the Web network</p> <p>Section 2.3 Network Security has been added.</p> <p>Section 2.7 and Appendix E have been added to describe the use of the Alabama Medicaid RAS.</p> <p>Section 3.1 shows the correct URLs for three environments that may be transitional and possible DNS error resolution steps</p> <p>Section 3.4 File Upload Message – X12 and Non-X12 Transactions has been added</p> <p>Section 3.5 – Updated the list of file types that can be downloaded.</p> <p>Section 3.7 – Updated list of system errors.</p> <p>Section 4 – Correct User IDs for Interactive Frame Relay</p>
11/04/2003	1.2	Susan Giannobile	<p>Section 2.3 – table 2.3 – URL name changed and removed acceptance test reference.</p> <p>Section 2.7 – URL name changed.</p> <p>Section 3.1 – table 3.1– URL name changed.</p>
07/27/2004	1.3	Susan Giannobile	<p>Section 3.7 – table 3.8 – Added Message Code 209 to 211.</p> <p>Appendix A – Updated the URL and File Type parameter.</p> <p>Appendix B – Updated the URL.</p> <p>Appendix C – Updated the URL.</p>

## 2 Introduction

### 2.1 Background

This document is intended for Software Vendors to use when developing applications to interact with the Alabama State Medicaid System. Processes to upload and download HIPAA compliant transaction batches via a secure Internet Web site and to submit interactive transaction via existing Frame Relay Lines are described and sample code is provided. This document also describes the process by which batches of NCPDP transactions can be uploaded for processing via the Web network. This functionality replaces the BBS upload and download processes. Currently, interactive transaction processing via the Internet is being developed and tested internally. Vendors who have voiced an interest in this technology will be contacted as soon as it is available for acceptance testing.

The ECS helpdesk is available to providers and vendors to answer questions, concerns, or to address any problems which may occur during transmission. The help desk hours are Monday through Friday 7:00 AM to 8:00 PM. Saturday (including holidays) 9:00 AM to 5:00 PM. The help desk can be reached in the following ways.

<u>Phone</u>	<u>Writing</u>	<u>E-mail</u>
(800) 456-1242	Electronic Data Systems (EDS)	
(334) 215-0111	Attn: ECS Help Desk	<a href="mailto:ecs@alixx.slg.eds.com">ecs@alixx.slg.eds.com</a>
(334) 215-4272 (fax)	301 Technacenter Drive Montgomery, AL 36117	

### 2.2 HIPAA Approved Transaction Processing

The State of Alabama is prepared to accept batch files in the HIPAA approved X12 and NCPDP 5.1 formats. Depending on the size of the submitted batch of transactions, approved responses to batch requests can be downloaded. In addition, following the payment adjudication cycle, unsolicited X12 responses can also be downloaded. See Table 2-1.

Table 2-1

Transaction Description	Request Transaction ID	Version Identifier	Response Transaction ID	Version Identifier
Eligibility Verification	270	004010X092A1	271	004010X092A1
Claim Status	276	004010X093A1	277	004010X093A1
Health Care Service - Prior Authorization	278	004010X094A1	278	004010X094A1
Managed Care Payment Order / Remittance Advice - Response only			820	004010X061A1
Managed Care Benefit Enrollment & Maintenance - Response only			834	004010X095A1
Institutional Claim	837	004010X096A1	CSR	Proprietary
Dental Claim	837	004010X097A1	CSR	Proprietary
Professional Claim	837	004010X098A1	CSR	Proprietary
Unsolicited Claim Status – Response only (277U)			27U	003070X070
Explanation of Payments – Response only			835	004010X091A1

Acknowledge Receipt – Response only			997	004010
NCPDP Pharmacy Claim	B1	5.1	NCP	
NCPDP Pharmacy Claim Reversal	B2	5.1	NCP	
NCPDP Eligibility Verification	E1	5.1	NCP	
Long Term Care Acceptance Report – Response only			LT1	
Long Term Care Rejected Report – Response only			LT2	

The transaction types in the following table may be entered interactively using the frame relay protocol described later in this document. See Table 2-2.

Table 2-2

Transaction Description	Request Transaction ID	Version Identifier	Response Transaction ID	Version Identifier
Eligibility Verification	270	004010X092A1	271	004010X092A1
Claim Status	276	004010X093A1	277	004010X093A1
NCPDP Pharmacy Claim	B1	5.1		
NCPDP Pharmacy claim reversal	B2	5.1		
NCPCP Eligibility Verification	E1	5.1		

## 2.3 Network Security

Before transactions can be processed within the Alabama Medicaid Automated Network, requester must obtain a Trading Partner ID and password from the ESC help desk. Contact information is documented in the Introduction, section 2 above. An initial password will be created and sent to the requester. Before beginning transmission, however, the password must be reset and maintenance questions and answers set up for future use. This process takes only a couple of minutes and is documented in the web users guide and help pages available at the web sites listed in Table 2-3 below. Since each environment owns a unique security database, security maintenance must be performed within each environment that is used. Additional Web functions authorized for use by the Trading Partner are listed on the menu panel on the left side of the window.

Table 2-3

Environment	Secure Internet URL
Production	<a href="https://almedicalprogram.alabama-medicaid.com/secure">https://almedicalprogram.alabama-medicaid.com/secure</a>
Model Office	<a href="https://almedicalprogram.alabama-medicaid.com/mod/secure">https://almedicalprogram.alabama-medicaid.com/mod/secure</a>

## 2.4 Web Interface

The current web interface is designed to support batch file uploads and downloads. There are two ways to use this interface. The first is to logon to the secure web site using a Trading Partner User ID and password as described in section 2.3 Network Security. This web site has web pages that allow users to upload and download files from and to directories within the user's PC or LAN. The second way is to use

a software program that runs on a user's PC or server that connects to the secure Trading Partner Web Services. The user's site sends a request using the HTTPS protocol containing parameters that include the Trading Partner User ID, the associated password, and the request data. The request data can include a request for a listing of files available for download, a specific file name to download, or a file to upload. The files can be transferred in compressed format or in standards ASCII text format. All data is transferred using the Secure Socket Layer (SSL) that encrypts the data over the network.

## **2.5 Client Software for Web Communications**

The client software can be written in any language that supports HTTPS for communicating with the Trading Partner Web Services. The request transactions are formatted in XML, but the data files transferred from and to the web services are in the HIPAA standard formats. The XML data is used to support the security and general interaction with the web services.

## **2.6 Client Software for Frame Relay Submissions**

Effective June 30, 2003, AT & T discontinued support for the X.25 line handler protocol, replacing it with a new service that uses IP addressing to mimic the X.25 process. Vendors currently transmitting NCPDP interactive transactions have been asked to change the dial-in phone number to begin using the new service. Please see section 4 for detailed specifications.

## **2.7 Use of the Alabama Medicaid RAS**

Trading Partners who do not have a contract with an Internet Service Provider (ISP) can optionally use a modem to dial into the Medicaid's Remote Access Server (RAS) to gain access to this web site only. If you live outside the Montgomery calling area, you must be able to place a long distance call over the phone line. An Internet browser will also be required to maintain your security ID and password. The EDS software is written to work best using the Internet Explorer Browser. This software is available to download from the Alabama Medicaid homepage at <http://www.alabama-medicaid.com> and from the Help Option on the secure HIPAA web site. Please see Appendix E for details on setting up a network connection using the Alabama Medicaid RAS.

# **3 Web Interface Specification**

## **3.1 Connecting to the server**

Before connecting to the web application, a network connection must first be established that provides access to the public internet. Optionally, a dial-up connection through the Alabama Remote Access Server (RAS) can be initiated. If your connection requires a proxy server to connect to the Internet this needs to be configurable in your application, the sample Java demonstrates this. Once connected, use the URLs in Table 3-1 to establish a connection with the applications. It is recommended that these URLs not be hard coded as part of your application, but remain configurable instead. The sample program provided in Java demonstrates how to establish a connection to these URLs. All of the transactions will require the connection be made using the Secure Socket Layer (SSL). The sample program provides an example of how to do this with Java. Other programming languages such as Visual Basic have other ways to connect using SSL, such as by using the WinInet libraries that are a part of Microsoft Windows.

As Table 3-1 indicates, several environments are available for sending transactions. The Model Office environment is intended for software testing as well as transaction compliance testing.

Table 3-1

Environment	Root URL

Production	almedicalprogram.alabama-medicaid.com
Model Office	almedicalprogram.alabama-medicaid.com/mod

To access the specific web applications append the URL suffixes as identified in Table 3-2.

Table 3-2

Web Application	URL Suffix
File Upload	/secure/WebUploadFromClient
File Download Specific Transaction Type	/secure/WebUploadNX12FromClient
Directory List	/secure/WebDirectoryDownloadFromClient
File Download	/secure/WebDownloadFromClient

In some situations, DNS connectivity problems may occur between the Alabama Medicaid secure web site and PCs or Computers on a LAN within a secure network. Below are some diagnostic steps that can be taken.

- 1) Click on Start -> Run...
- 2) In the "Open:" box type: "CMD" or "command" depending on the operating system.
- 3) In the command prompt type: nslookup
- 4) At the > prompt type: almedicalprogram.alabama-medicaid.com
- 5) One of two results should happen
  - a) A message will be displayed saying the dns server cannot find the address for example:  
"\*\*\*\*dns1.alxix.hcg.eds.com can't find almedicalprogram.alabama-medicaid.com"  
"Non-existent domain".
  - b) Non-authoritative answer:  
Name: almedicalprogram.alabama-medicaid.com  
Address: 65.222.80.203
- 6) If you receive "a)" above, contact your network administrators due to DNS problems.
- 7) If you receive "b)" above and still cannot connect to the site, contact your network administrators, because your network probably has a proxy server configured for http and https.

An entry may need to be added for almedicalprogram.alabama-medicaid.com (65.222.80.203) address into your internal DNS. Or, everyone within the network who needs to access the Medicaid secure site may need to add the same name and ip address to their lmhosts file.

## 3.2 HTTPS Messages

In order to interact with the web applications once a connection is established standard messages must be sent. The body of the message is formatted in simple XML. The XML messages are described in detail in the following sections. The message for uploading files uses the standard HTTP Multipart MIME type format. The Multipart MIME type format is the same format that is used to upload files through a web browser. All the other messages use a text XML format. Except for downloading a file all responses are in XML format. This XML should be parsed using standard XML parsing routines.

## 3.3 File Upload Message – X12 Transactions only

The file upload message must be in the HTTPS Multipart MIME type format. There are two parts to the message. The first part is user identification data in XML format and the second is the actual file being uploaded. The file being uploaded must be in an ASCII text format or zip format. The response to this message is a General Response Message. See the "General Response Message" section for details.

When using the Multipart form type data is separated by a unique string of characters that serves as the boundary between the parts of data. This number must be unique and not a part of the actual data uploaded as part of the request. In the multipart mime type HTTP header the boundary characters are defined, see sample request. The first part of the request is an XML message which includes security information used to authenticate the user has access to perform the upload. See Table 3-3. A sample of the XML request and it's associated schema are presented on the following pages. Following the XML part is the data file being uploaded. No more than one file can be uploaded at a time and the total length of the entire upload message cannot exceed 8 megabytes or the whole transaction will be rejected. Additionally, batches containing more than 5000 claims or inquiry transactions will be held for processing until off-peak, thus giving priority to interactive and small batches.

See Appendix A for sample upload program.

URL: <https://<Root URL>/secure/WebUploadFromClient>

Table 3-3 File upload X-12 XML Message Description

<b>Element</b>	<b>Description</b>	<b>Length</b>	<b>Required</b>
TradingPartnerId	The Trading Partner ID is used to identify the transaction submitter. The ID, along with the User ID and Password, is used to authenticate the user. Sample Data: 22222222 Format: Alphanumeric	3 - 35	Y
UserId	The User ID is used to authenticate the user submitting the request. The User ID must be authorized to submit for the Trading Partner ID. Sample Data: xyzuser Format: Alphanumeric	3-15	Y
Password	The Password is used in conjunction with the User ID to ensure the validity of the user making the request. Sample Data: jmstp567 Format: Alphanumeric	6-8	Y
Function	This is the Upload file function name. This value must always be filled with "UPLOADFILE" to be a valid request. Required value: UPLOADFILE Format: Alphanumeric	10	Y
FileName	The path and name of the file being uploaded. Sample Data: d:\myfile.dat Format: Alphanumeric	1 - 256	Y
FileSize	Specifies the size of the file. The size is compared against the actual size of the file uploaded to ensure no data is lost. If the file is in a zipped format, this is the zipped file size. Sample Data: 1022 Format: Numeric	8	Y

Sample HTTP multipart file upload request:

```
POST /secure/WebUploadFromClient HTTP/1.0
Content-Type: multipart/form-data; boundary=7d021a37605f0
User-Agent: Java1.2.2
Host: www.almedicalprogram.alabama-medicaid.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-length: 698

--7d021a37605f0
Content-Disposition: form-data; name="message"

<?xml version="1.0" encoding="UTF-8"?>
<UploadRequest>
  <IdentificationHeader>
    <TradingPartnerId>222222222</TradingPartnerId>
    <UserId>xyzuser</UserId>
    <Password>jmstp567</Password>
  </IdentificationHeader>
  <Transaction>
    <Function>uploadFile</Function>
    <FileName>d:\myfile.dat</FileName>
    <FileSize>45</FileSize>
  </Transaction>
</UploadRequest>
--7d021a37605f0
Content-Disposition: form-data;name="textFileAttached"; filename="d:\temp\test.txt"
Content-Type: text/plain

File in X12N format to upload.
data2
data3.
--7d021a37605f0
```

**File Upload Request XML Schema:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="FileName" type="xsd:string"/>
  <xsd:element name="FileSize" type="xsd:string"/>
  <xsd:element name="Function" type="xsd:string"/>
  <xsd:element name="IdentificationHeader">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="TradingPartnerId" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="UserId" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Password" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Password" type="xsd:string"/>
  <xsd:element name="TradingPartnerId" type="xsd:string"/>
  <xsd:element name="Transaction">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="Function" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="FileName" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="FileSize" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="UploadRequest">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="IdentificationHeader" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Transaction" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="UserId" type="xsd:string"/>
</xsd:schema>
```

### 3.4 File Upload Message – X12 and Non-X12 Transactions

The functionality and syntax of this upload method is exactly the same as the process defined and illustrated in section 3.3 except for the addition of the file type element. By entering one of the three allowed file types, X12, NCP or LTC, the Web application can direct non-X12 transactions to a different destination for processing. If you are currently using the technique described in 3.4 to upload X12 transactions, you do not need to alter your connection logic.

The file upload message must be in the HTTPS Multipart MIME type format. There are two parts to the message. The first part is user identification data in XML format and the second is the actual file being uploaded. The file being uploaded must be in an ASCII text format or zip format. The response to this message is a General Response Message. See the “General Response Message” section for details.

When using the Multipart form type data is separated by a unique string of characters that serves as the boundary between the parts of data. This number must be unique and not a part of the actual data uploaded as part of the request. In the multipart mime type HTTP header the boundary characters are defined. See sample request. The first part of the request is an XML message which includes security information used to authenticate the user has access to perform the upload. See Table 3-4. A sample of the XML request and its associated schema are presented on the following pages. Following the XML request, is the data file being uploaded. No more than one file can be uploaded at a time and the total length of the entire upload message cannot exceed 8 megabytes or the whole transaction will be rejected. Additionally, batches containing more than 5000 claims or inquiry transactions will be held for processing until off-peak, thus giving priority to interactive transactions and small batches.

See Appendix A for sample upload program. Please note that this example is specifically for the technique described in section 3.3. Minor modifications are required to produce the XML message for Non-12 Transactions.

URL: <https://<Root URL>/secure/WebUploadNX12FromClient>

Table 3-4 File upload Non-X12 XML Message Description

Element	Description	Length	Required
TradingPartnerId	The Trading Partner ID is used to identify the transaction submitter. The ID, along with the User ID and Password, is used to authenticate the user. Sample Data: 22222222 Format: Alphanumeric	3 - 35	Y
UserId	The User ID is used to authenticate the user submitting the request. The User ID must be authorized to submit for the Trading Partner ID. Sample Data: xyzuser Format: Alphanumeric	3-15	Y
Password	The Password is used in conjunction with the User ID to ensure the validity of the user making the request. Sample Data: jmstpl567 Format: Alphanumeric	6-8	Y
Function	This is the Upload file function name. This value must always be filled with “UPLOADFILE” to be a valid request. Required value: UPLOADFILE Format: Alphanumeric	10	Y
FileName	The path and name of the file being uploaded. Sample Data: d:\myfile.dat Format: Alphanumeric	1 - 256	Y

<b>Element</b>	<b>Description</b>	<b>Length</b>	<b>Required</b>
FileType	Specifies the type of the file to be uploaded. Following is a list of file types being used in Alabama. X12 - X12 Format NCP - NCPDP LTC - Long Term Care Format: Alphanumeric	3	Y
FileSize	Specifies the size of the file. The size is compared against the actual size of the file uploaded to ensure no data is lost. If the file is in a zipped format, this is the zipped file size. Sample Data: 1022 Format: Numeric	8	Y

Sample HTTP multipart file upload request:

```

POST /secure/WebUploadFromClient HTTP/1.0
Content-Type: multipart/form-data; boundary=7d021a37605f0
User-Agent: Java1.2.2
Host: www.almedicalprogram.alabama-medicaid.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-length: 698

--7d021a37605f0
Content-Disposition: form-data; name="message"

<?xml version="1.0" encoding="UTF-8"?>
<UploadRequest>
  <IdentificationHeader>
    <TradingPartnerId>22222222</TradingPartnerId>
    <UserId>xyzuser</UserId>
    <Password>jmstp567</Password>
  </IdentificationHeader>
  <Transaction>
    <Function>uploadFile</Function>
    <FileName>d:\myfile.dat</FileName>
    <FileType>ltc</FileType>
    <FileSize>45</FileSize>
  </Transaction>
</UploadRequest>
--7d021a37605f0
Content-Disposition: form-data;name="textFileAttached"; filename="d:\temp\test.txt"
Content-Type: text/plain

File in standard X12, NCPDP 5.1 or LTC format to upload.
data2
data3.
--7d021a37605f0

```

**File Upload Request XML Schema:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="FileName" type="xsd:string"/>
  <xsd:element name="FileSize" type="xsd:string"/>
  <xsd:element name="Function" type="xsd:string"/>
  <xsd:element name="IdentificationHeader">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="TradingPartnerId" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="UserId" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Password" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Password" type="xsd:string"/>
  <xsd:element name="TradingPartnerId" type="xsd:string"/>
  <xsd:element name="Transaction">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="Function" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="FileName" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="FileType" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="FileSize" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="UploadRequest">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="IdentificationHeader" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="Transaction" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="UserId" type="xsd:string"/>
</xsd:schema>
```

### 3.5 Directory List Message

The directory list message is an XML formatted message sent over HTTPS. The directory list function is used to get the name(s) of available files for downloading. A specific file name must be specified in the download message in order to get the file. There are several options that can be specified in the XML message to filter what is returned in the directory list. See Table 3-5. There are two possible return type messages. If the directory listing is successful a directory listing response message is returned. See Table 3-5. The directory listing response message contains all the files meeting the criteria specified in the request message. If the directory listing request message is incorrectly formatted or there are problems processing the request a General Response message is returned with a description of the error. See the “General Response Message” section for details.

See Appendix B for directory list sample program.

URL: <https://<Root URL>/secure/WebDirectoryDownloadFromClient>

Table 3-5 Directory Listing Request Message Description

Element	Description	Length	Required
TradingPartnerId	The Trading Partner Id is used to identify the transaction submitter. The ID is used to limit the list of files in the directory list to only those belonging to this Trading Partner. The Trading Partner ID is cross validated against the User ID and the User ID must be authorized to submit requests for this Trading Partner ID. Sample Data: 222222222 Format: Alphanumeric	3 - 35	Y
UserId	The User ID is used to authenticate the user submitting the request. The User ID must be authorized to submit for the Trading Partner ID. Sample Data: xyzuser Format: Alphanumeric	3-15	Y
Password	The Password is used in conjunction with the User ID to ensure the validity of the user making the request. Sample Data: jmstpl567 Format: Alphanumeric	6-8	Y
Function	This is the Directory List function name. This value must always be filled with “DIRLIST” to be a valid request. Required value: DIRLIST Format: Alphanumeric	7	Y
FilesToReturnCount	Specifies the maximum number of files to return in the directory list. If not included, then all files are returned. Sample Data: 50 Format: Numeric	0-6	N

<b>Element</b>	<b>Description</b>	<b>Length</b>	<b>Required</b>
FileType	<p>This indicates the types of files to be returned in the directory list. For example, if a file type of 997 is entered, the list will only include files of functional acknowledgments. To request a list containing 997s and 835 RAs, specify two file types - one with 997 and the other with 835. Up to 10 file types can be specified at a time. If no file types are specified, then all types are returned.</p> <p>Following is a list of available file types .</p> <ul style="list-style-type: none"> <li>271 – Eligibility Response</li> <li>277 – Claim Status Response</li> <li>278 –Health Care Service Review Response</li> <li>820 – Health Plan Payment</li> <li>834 – Benefit Enrollment and Maintenance</li> <li>835 – Remittance Advice</li> <li>27U - Unsolicited Claim Status</li> <li>997 – Functional Acknowledgment</li> <li>CSR - Claim Submission Response</li> <li>LT1 - Long Term Care (Accepted)</li> <li>LT2 - Long Term Care (Rejected)</li> <li>NCP - NCPDP</li> </ul> <p>Format: Alphanumeric, occurs up to 10 times – UPPER CASE</p>	3	N
FileStatus	Indicator to select files based on the following: A = All files N = New files only D = Previously downloaded files only	1	Y
FilesCreatedFromDate	<p>The file creation from date is used as a filter to only return files in the directory list which are greater than or equal to this date. If this element is not specified, all files are listed, up through the FilesCreatedToDate if specified.</p> <p>Sample Data: 20020701</p> <p>Format: 4 digit year, 2 digit month, 2 digit day (CCYYMMDD)</p>	8	N
FilesCreatedToDate	<p>The file creation to date is used as a filter to only return files in the directory list which are less than or equal to this date. If this element is not specified, all files are listed, equal to and greater than the FilesCreatedFromDate if specified. If neither the from or to date element is specified, all files are returned in the directory list.</p> <p>Sample Data: 20020731</p> <p>Format: 4 digit year, 2 digit month, 2 digit day (CCYYMMDD)</p>	8	N

Sample HTTP Request for Directory Listing:

```
POST /secure/WebDirectoryDownloadFromClient HTTP/1.0
Content-Type: text/xml
User-Agent: Java1.2.2
Host: www.almedicalprogram.alabama-medicaid.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-length: 708

<?xml version="1.0" encoding="UTF-8"?>
<DirectoryRequest>
  <IdentificationHeader>
    <TradingPartnerId>22222222</TradingPartnerId>
    <UserId>xyzuser</UserId>
    <Password>jmstp567</Password>
  </IdentificationHeader>
  <Transaction>
    <Function>DIRLIST</Function>
    <FilesToReturnCount>50</FilesToReturnCount>
    <SelectedFileTypes>
      <FileType>997</FileType>
      <FileType>835</FileType>
    </SelectedFileTypes>
    <FileStatus>A</FileStatus>
    <FilesCreatedFromDate>20020102</FilesCreatedFromDate>
    <FilesCreatedToDate>20020103</FilesCreatedToDate>
  </Transaction>
</DirectoryRequest>
```

## Directory Listing Request XML Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0" xml:lang="EN">
    <xsd:element name="DirectoryRequest">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="IdentificationHeader" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Transaction" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="TradingPartnerId" type="xsd:string"/>
    <xsd:element name="UserId" type="xsd:string"/>
    <xsd:element name="Password" type="xsd:string"/>

    <xsd:element name="IdentificationHeader">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="TradingPartnerId" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="UserId" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Password" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Function" type="xsd:string"/>
    <xsd:element name="FilesToReturnCount" type="xsd:string"/>
    <xsd:element name="FileType" type="xsd:string"/>
    <xsd:element name="SelectedFileTypes">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="FileType" minOccurs="0" maxOccurs="10"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="FilesStatus" type="xsd:string"/>
    <xsd:element name="FilesCreatedFromDate" type="xsd:string"/>
    <xsd:element name="FilesCreatedToDate" type="xsd:string"/>

    <xsd:element name="Transaction">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="Function" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FilesToReturnCount" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="SelectedFileTypes" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FilesStatus" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="FilesCreatedFromDate" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="FilesCreatedToDate" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

Table 3-6 Directory Listing Response Message Description

<b>Element</b>	<b>Description</b>	<b>Length</b>	<b>Required</b>
TradingPartnerId	The Trading Partner ID from the request. Sample Data: 22222222 Format: Alphanumeric	3 - 35	Y
UserId	The User ID from the request. Sample Data: xyzuser Format: Alphanumeric	3-15	Y
Function	The function from the request. Required value: DIRRESP Format: Alphanumeric	7	Y
FilesReturnedCount	The number of files returned in the directory listing. Sample Data: 50 Format: Numeric	1 - 6	Y
FileName	The file name given to the file. This is a system generated name. This is the same name needed to request the file for download. Sample Data: 2222222220020702001 Format: Alphanumeric	9 - 40	Y
FileCreationDate	The date the file was created. Sample Data: 20020701 Format: 4 digit year, 2 digit month, 2 digit day (CCYYMMDD)	8	Y
FileType	This is the type of file. The file type must be coded in UPPER CASE. Sample Data: 997	3	Y
FileSize	The size in bytes of the file in unzipped format. Sample Data: 50000 Format: Numeric	2 - 8	Y
LastDownloadDate	The date the file was last downloaded. If the file has not been downloaded this is empty. Sample Data: 20020701 Format: 4 digit year, 2 digit month, 2 digit day (CCYYMMDD)	8	N
LastDownloadUserId	This is the User ID of the last user to download the file. This is empty if the file has not been downloaded.	3-15	N

Sample HTTP Response from Directory Listing Request:

```
HTTP/1.1 200 OK
Date: Mon, 05 Aug 2002 17:39:13 GMT
Server: IBM HTTP Server/1.3.12.3 Apache/1.3.12 (Win32)
Last-Modified: Fri, 26 Jul 2002 21:24:30 GMT
Etag: "0-2dae-3d41be0e"
Accept-Ranges: bytes
Content-Length: 11694
Connection: close
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<DirectoryResponse>
  <IdentificationHeader>
    <TradingPartnerId>22222222</TradingPartnerId>
    <UserId>xyzuser</UserId>
  </IdentificationHeader>
  <Transaction>
    <Function>DIRLIST</Function>
    <FilesReturnedCount>2</FilesReturnedCount>
    <DirectoryList>
      <FileName>2222222220020702001</FileName>
      <FileCreationDate>20020702</FileCreationDate>
      <FileType>997</FileType>
      <FileSize>121</FileSize>
      <LastDownLoadDate></LastDownLoadDate>
      <LastDownLoadUserId></LastDownLoadLastUserId>
    </DirectoryList>
    <DirectoryList>
      <FileName>2222222220020702002</FileName>
      <FileCreationDate>20020702</FileCreationDate>
      <FileType>835</FileType>
      <FileSize>2017</FileSize>
      <LastDownLoadDate>20020709</LastDownLoadDate>
      <LastDownLoadUserId>xyzuser</LastDownLoadLastUserId>
    </DirectoryList>
  </Transaction>
</DirectoryResponse>
```

## Directory Listing Response XML Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="DirectoryList">
        <xsd:complexType>
            <xsd:sequence minOccurs="0" maxOccurs="450">
                <xsd:element ref="FileName" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FileCreationDate" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FileType" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FileSize" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="LastDownLoadDate" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="LastDownLoadUserId" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="DirectoryResponse">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="IdentificationHeader" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Transaction" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="FileCreationDate" type="xsd:string"/>
    <xsd:element name="FileName" type="xsd:string"/>
    <xsd:element name="FileSize" type="xsd:string"/>
    <xsd:element name="LastDownLoadDate" type="xsd:string"/>
    <xsd:element name="LastDownLoadUserId" type="xsd:string"/>
    <xsd:element name="FileType" type="xsd:string"/>
    <xsd:element name="FilesReturnedCount" type="xsd:string"/>
    <xsd:element name="Function" type="xsd:string"/>
    <xsd:element name="IdentificationHeader">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="TradingPartnerId" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="UserId" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="TradingPartnerId" type="xsd:string"/>
    <xsd:element name="Transaction">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="Function" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FilesReturnedCount" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="DirectoryList" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="UserId" type="xsd:string"/>
</xsd:schema>
```

### 3.6 Download a File Message

The download a file message is an XML formatted message sent over HTTPS. A file name from the directory listing response is used to request the file to download. This file name must be specified in the download message in order to get the file. The request is made using XML. See Table 3-7 for details. There are two possible returns from a download request. If the file requested was found and there was no problems processing the request then the data file is returned. If any problems occurred processing the request then a General Response Message is returned. The client application must recognize the difference between a file being returned and the General Response Message. See the "General Response Message" section for details.

See Appendix C for download sample program.

URL: <https://<Root URL>/secure/WebDownloadFromClient>

Table 3-7 Download a file Message Description

Element	Description	Length	Required
TradingPartnerId	The Trading Partner ID is used to identify the transaction submitter. The Trading Partner ID is cross validated against the User ID and the User ID must be authorized to submit requests for this Trading Partner ID. Sample Data: 22222222 Format: Alphanumeric	3 - 35	Y
UserId	The User ID is used to authenticate the user submitting the request. The User ID must be authorized to submit for the Trading Partner ID. Sample Data: xyzuser Format: Alphanumeric	3-9	Y
Password	The Password is used in conjunction with the User ID to ensure the validity of the user making the request. Sample Data: jmstp567 Format: Alphanumeric	6-8	Y
Function	This is the File Download function name. This value must always be filled with "DOWNLOAD" to be a valid request. Required value: DOWNLOAD Format: Alphanumeric	8	Y
FileName	Specifies the name of the file to download. The name can be found from the directory list request. Sample Data: 000000123 Format: Alphanumeric	9 - 40	N
FileFormat	Specifies the file format you want the file downloaded in. Valid Values are: TXT – Requests the file in standard ASCII Text format. ZIP – Requests the file in zipped format.	3	Y

**Sample HTTP Request for a file download:**

```

POST /secure/WebDownloadFromClient HTTP/1.0
Content-Type: text/xml
User-Agent: Java1.2.2
Host: www.almedicalprogram.alabama-medicaid.com
Accept: text/html, image/gif, image/jpeg, *, */*, q=.2, */*; q=.2
Content-length: 359
<?xml version="1.0" encoding="UTF-8"?>
<DownloadRequest>
    <IdentificationHeader>
        <TradingPartnerId>222222222</TradingPartnerId>
        <UserId>xyzuser</UserId>
        <Password>jmstp567</Password>
    </IdentificationHeader>
    <Transaction>
        <Function>DOWNLOAD</Function>
        <FileName>000000123</FileName>
        <FileFormat>TXT</FileFormat>
    </Transaction>
</DownloadRequest>

```

**Download file Request XML Schema**

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="DownloadRequest">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="IdentificationHeader" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Transaction" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="FileName" type="xsd:string"/>
    <xsd:element name="FileFormat" type="xsd:string"/>
    <xsd:element name="Function" type="xsd:string"/>
    <xsd:element name="IdentificationHeader">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="TradingPartnerId" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="UserId" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="Password" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Password" type="xsd:string"/>
    <xsd:element name="TradingPartnerId" type="xsd:string"/>
    <xsd:element name="Transaction">
        <xsd:complexType>
            <xsd:sequence minOccurs="1" maxOccurs="1">
                <xsd:element ref="Function" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FileName" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="FileFormat" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="UserId" type="xsd:string"/>
</xsd:schema>

```

### 3.7 General Response Message

The General Response Message can be returned for any type request message. This response message indicates whether a request was successful or not. Some requests such as the directory request do not return a general response message, but instead a directory listing response message is returned. If however there is problem returning the directory listing response a General Response Message will be returned instead.

Table 3-8 File Upload Response Message Description

Element	Description	Length	Required
MessageCode	<p>Code identifying the message. Sample Data: 100 Format: Numeric</p> <p>Currently there are only two defined codes. 0 – Successful 202 – Security authentication failed 203 – Security password expired 204 – Security password suspended 205 – Security authorization failed 206 – Not a trading partner 207 – Zero byte file 208 – No file 209 – Invalid File Type 210 – Trading partner missing 211 – Trading partner mismatch 999 – Unable to process request</p>	1 - 3	Y
MessageDesc	<p>Descriptive message. Sample Data: File was successfully uploaded. Format: Alphanumeric</p>	Up to 256	Y
MessageType	<p>Code identifying the type of message this is. Sample Data: A. Format: Alpha Codes: A – Transaction Accepted with no errors. W – Transaction failed, see message for more information and retry. E – Fatal error.</p>	1	Y
ControlNumber	<p>This is the transaction control number. This is currently only used by Upload Transactions responses. Format: Numeric</p>	1 – 18	N

Sample HTTP Response Message:

```

HTTP/1.1 200 OK
Date: Mon, 05 Aug 2002 17:39:13 GMT
Server: IBM HTTP Server/1.3.12.3 Apache/1.3.12 (Win32)
Last-Modified: Fri, 26 Jul 2002 21:24:30 GMT
ETag: "0-2dae-3d41be0e"
Accept-Ranges: bytes
Content-Length: 11694
Connection: close
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<ResponseMessage>
    <Messages>
        <MessageCode>0</MessageCode>
        <MessageDesc>Successful Upload</MessageDesc>
        <MessageType>A</MessageType>
        <ControlNumber>12345</ControlNumber>
    </Messages>
</ResponseMessage>

```

THIS DOCUMENT IS SUBJECT TO CHANGE

---

Directory Listing Response XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="MessageCode" type="xsd:string"/>
  <xsd:element name="MessageDesc" type="xsd:string"/>
  <xsd:element name="MessageType" type="xsd:string"/>
  <xsd:element name="ControlNumber" type="xsd:string"/>
  <xsd:element name="Messages">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="MessageCode" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="MessageDesc" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="MessageType" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="ControlNumber" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="ResponseMessage">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Messages" minOccurs="1" maxOccurs="10"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

## 4 Frame Relay Interface Specification

### AT&T Asynchronous Dial Service

AT&T WorldNet Asynchronous Service (WNAS) supports asynchronous dial-in speeds of 14400, 9600, 4800, 2400, and 1200 bps. Modem protocols supported are V.32bis, V.32, V.22bis, V.22, Bell212a, and Bell103 modem standards with V.42bis, V.42, V.90, and MNP protocols 1-5.

#### To dial into WNAS:

##### 1. Modem Setup

You can dial into WNAS using a variety of speeds and modem settings as mentioned above. Data compression can be enabled or disabled. Error correction can be enabled or disabled, however, we do not recommend connection without error correction enabled. WNAS supports 8-1-N or 7-1-E data-stop-parity settings.

##### 2. Dialing WNAS

WNAS nationwide telephone number is:

**Nationwide Service: 866-627-0017.**

Use this nationwide toll-free service number that is accessible from any location in the country.

End-users should be provided the capability to change the telephone number.

##### 3. Send User ID

Provider Type	WNAS User Ids
Production	ALHP0 (zero)
Model Office	ALHM
Test	ALHT0 (zero)

Script actions required to connect to WNAS

Script Action	Network Response	What's Happening
Send a single <CR> (ASCII hex value 'OD') immediately upon receiving a positive modem connect	WNAS welcome message and request for USER-ID. 'Sign-on:' prompt displays.	WNAS waiting for USER-ID
Send assigned USER-ID followed by <CR>.	Connection to application is made. WNAS responds with <ENQ> (ASCII value '05').	Connected to OLTP and ready to send a packet.

Note: The <ENQ> response indicates that a direct connection to the OLTP has been established. When the transaction is complete, the AEVCS host and WNAS network sessions are immediately disconnected.

A communication package such as Procomm Plus or Crosstalk can be used to verify the script process. Setup the package to dial the desired number, type in each script action, and wait for the appropriate response from the network. A hang-up signal must be initiated after receiving the <ENQ> response.

## Appendix A – Sample Upload a File Program

### Upload a file program example

```

package com.eds.ne.regional.samples;

import com.sun.net.ssl.HttpsURLConnection;
import com.sun.net.ssl.HostnameVerifier;
import java.net.*;
import java.io.*;

/**
 * This program is provided as sample only as is without warranty or
 * condition of any kind, either expressed or implied, including, but
 * not limited to, the implied warranties of merchantability and
 * fitness for a particular purpose.
 *
 * This sample File Upload program connects to the web server then
 * sends it a HTTP Multi-Part message containing an XML message and
 * the file data to be uploaded. The name of the file to be uploaded
 * is passed to the program, along with the server name. See the
 * acceptArguments method for parameters to pass on the command line.
 * If no parameters are passed the program will prompt the user for
 * the input parameters then execute. The upload response is simply
 * displayed in standard out. This program differs from the other
 * sample programs as it builds the XML message in the code rather
 * than reading it from a file.
 *
 * See the vendor specifications for details of the format of the XML message
 * for a file upload.
 *
 * <P>
 * In order to negotiate the SSL certificate of the web server the client has
 * to have a trusted certificate in its Java keystore file. The web sites use
 * GeoTrust Certificates. If this Certificate is not in your Java installations
 * keystore you will need to install it. To install the certificate you can use
 * the java keytool command
 *
 * This sample program was written using JDK 1.2.2. In order for the sample program
 * to communicate over SSL the Java Secure Socket Extension (JSSE) needs to be installed.
 * See http://java.sun.com/products/jsse/INSTALL.html.
 * After installing JSSE add the following jar files must be in the class path before
 * running the sample program: drive:/JSSE/jsse.jar;drive:/JSSE/jnet.jar;drive:/JSSE/jcert.jar
 * </P>
 * <HR>
 * <P>
 * <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0" WIDTH="100%">
 * <TR BGCOLOR="#9393ff">
 *   <TD COLSPAN=4><FONT SIZE="+2"><B>Modification History</B></FONT></TD>
 * </TR>
 * <TR BGCOLOR="white" CLASS="TableRowColor">
 *   <TH>Reason</TH><TH>Date</TH><TH>Systems Engineer</TH><TH>Description of Modification</TH>
 * </TR>
 * <TR>
 *   <TD ALIGN=LEFT>Initial deployment of this class.</TD>
 *   <TD ALIGN=RIGHT>12/19/2002</TD>
 *   <TD ALIGN=RIGHT>EDS</TD>
 *   <TD>Initial deployment of this class.</TD>
 * </TR>
 * </TABLE>
 *
 * @author Michael Smith
 *
 * @see SSLTunnelSocketFactory
 */
public class ClientFileUpload
{
/**
 * Users file Upload User Id. This user id is authenticated
 * on the server before uploading the users data.
 */
}

```

```

        String userId = "";
/**
 * Upload User Password.
 */
        String password = "";
/**
 * Upload File drive:\path\name.
 */
        String fileName = "d:\\temp\\test.txt";
/**
 * Upload File Type. See Table 3-4 for detail.
 */
        String fileType = "X12";
/**
 * Upload File size.
 */
        String fileSize = "0";
/**
 * Users proxy servers name or IP address.
 */
        String proxyServer = null;
/**
 * Users Proxys Port assignment.
 */
        String proxyPort = "80";
/**
 * True if Users sets their proxy server.
 */
        boolean thereIsProxy = false;
/**
 * Users Proxy Server User Id.
 */
        String proxyUser = null;
/**
 * Users proxy server password.
 */
        String proxyPassword = null;
/**
 * URL to post the data to. Defaults to test Web Server.
 */
        String postUrl = "https://almedicalprogram.alabama-
medicaid.com/mod/secure/WebUploadNX12FromClient";
/**
 * HTTP Protocol Switch used to turn on HTTP only uploads.
*/
        boolean httpSwitch = false;

        final static String XML_START = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
        final static String XML_UPLOAD_MESSAGE_START = "<UploadRequest>";
        final static String XML_UPLOAD_MESSAGE_END   = "</UploadRequest>";
        final static String XML_IDENTIFICATION_START = "<IdentificationHeader>";
        final static String XML_IDENTIFICATION_END   = "</IdentificationHeader>";
        final static String XML_TRADING_PARTNER_ID_START = "<TradingPartnerId>";
        final static String XML_TRADING_PARTNER_ID_END   = "</TradingPartnerId>";
        final static String XML_USER_ID_START = "<UserId>";
        final static String XML_USER_ID_END   = "</UserId>";
        final static String XML_PASSWORD_START = "<Password>";
        final static String XML_PASSWORD_END   = "</Password>";

        final static String XML_TRANSACTION_START = "<Transaction>";
        final static String XML_TRANSACTION_END   = "</Transaction>";
        final static String XML_FUNCTION_START = "<Function>";
        final static String XML_FUNCTION_END   = "</Function>";
        final static String XML_FILE_NAME_START = "<FileName>";
        final static String XML_FILE_NAME_END   = "</FileName>";
        final static String XML_FILE_TYPE_START = "<FileType>";
        final static String XML_FILE_TYPE_END   = "</FileType>";
        final static String XML_FILE_SIZE_START = "<FileSize>";
        final static String XML_FILE_SIZE_END   = "</FileSize>";

/**
 * ClientFileUpload default constructor calls it's super, then
 * setups System properties for SSL protocol.
 */
public ClientFileUpload()

```

```

{
    super();

    System.setProperty(
        "java.protocol.handler.pkgs",
        "com.sun.net.ssl.internal.www.protocol");

    // Required for ssl support and avoiding error:
    // "unknown protocol: https" exception
    java.security.Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
}
/***
 * Arguments passed on the command line are validated and
 * set by this method. The parameters are as follows:
 * <pre>
 * UserID Password FileName [URL] [proxyServer ProxyUser ProxyPassword]
 * </pre>
 *
 * @exception Exception - Invalid arguments message.
 */
public void acceptArgs(String[] args)
    throws Exception
{
    if (args.length >= 3)
    {
        userId          = args[0];
        password        = args[1];
        fileName        = args[2];
    }

    setFileSize();

    if (args.length == 4)
    {
        postUrl         = args[3];
    }

    if (args.length == 6)
    {
        proxyServer     = args[3];
        proxyUser       = args[4];
        proxyPassword   = args[5];
        thereIsProxy   = true;
    }

    if (args.length == 7)
    {
        postUrl         = args[3];
        proxyServer     = args[4];
        proxyUser       = args[5];
        proxyPassword   = args[6];
        thereIsProxy   = true;
    }

    if (args.length == 3 || args.length == 4 || args.length == 6 || args.length == 7)
    {
        // continue
    }
    else
    {
        throw new Exception("Invalid number of parameters found:" + args.length + " must
be 3, 4, 6, or 7 \n" +
                           "Valid Params are: UserID Password FileName [URL] [proxyServer
ProxyUser ProxyPassword]");
    }
}
/***
 * Build the header information needed for the multi-part format before actually
 * sending the file data. Supports both zipped files and text file formats.
 *
 * @param streamSend StringBuffer
 */
public void buildUploadFileHeader(StringBuffer streamSend)
{

```

```

// Each file is also sent within a boundary delimiter

if (fileName.endsWith(".zip"))
{
    // Zipped file format
    streamSend
        .append("Content-Disposition: form-data;name=\"zipFileAttached\"; filename=\"")
        .append(fileName)
        .append("\r\n")
        .append("Content-Type: application/x-zip-compressed\r\n")
        .append("\r\n");
}
else
{
    // Or Plain Text File
    streamSend
        .append("Content-Disposition: form-data;name=\"textFileAttached\"; filename=\"")
        .append(fileName)
        .append("\r\n");
    streamSend.append("Content-Type: text/plain\r\n");
    streamSend.append("\r\n");
}
}

/**
 * Builds the necessary format to pass parameter values with the upload file.
 *
 * @param streamSend StringBuffer
 * @param paramName java.lang.String
 * @param paramValue java.lang.String
 * @param boundry java.lang.String
 */
public void buildUploadParam(StringBuffer streamSend, String paramName, String paramValue, String boundary)
{
    streamSend
        .append("Content-Disposition: form-data; name=\"\"")
        .append(paramName)
        .append("\r\n\r\n")
        .append(paramValue)
        .append("\r\n")
        .append("--" + boundary + "\r\n");
}

/**
 * Builds the XML required in the Upload Message.
 *
 * @return java.lang.String
 */
public String createXmlMessage()
{
    StringBuffer xmlMessage = new StringBuffer()
        .append(XML_START)
        .append(XML_UPLOAD_MESSAGE_START)

        .append(XML_IDENTIFICATION_START)
        .append(XML_TRADING_PARTNER_ID_START)
        .append(userId)
        .append(XML_TRADING_PARTNER_ID_END)
        .append(XML_USER_ID_START)
        .append(userId)
        .append(XML_USER_ID_END)
        .append(XML_PASSWORD_START)
        .append(password)
        .append(XML_PASSWORD_END)
        .append(XML_IDENTIFICATION_END)

        .append(XML_TRANSACTION_START)
        .append(XML_FUNCTION_START)
        .append("uploadFile")
        .append(XML_FUNCTION_END)
        .append(XML_FILE_NAME_START)
        .append(fileName)
        .append(XML_FILE_NAME_END)
        .append(XML_FILE_TYPE_START)
        .append(fileType)
        .append(XML_FILE_TYPE_END)
}

```

```

.append(XML_FILE_SIZE_START)
.append(fileSize)
.append(XML_FILE_SIZE_END)
.append(XML_TRANSACTION_END)
.append(XML_UPLOAD_MESSAGE_END);

    return xmlMessage.toString();
}
/**
 * Main creates an instance of ClientFileUpload then
 * calls it's runProcess method.
 *
 * @param args java.lang.String[]
 */
public static void main(String[] args)
{
    ClientFileUpload clientFileUpload = new ClientFileUpload();

    clientFileUpload.setHTTP();

    clientFileUpload.runProcess(args);

}
/**
 * Override Hostname Verifier to ignore Certificate errors where the
 * URL on the certificate doesn't match the URL being accessed.
 * This should only be done for testing.
 *
 * @param theURLConnection URLConnection
 */
public void overrideHostNameVerifier(URLConnection theURLConnection)
{
    System.out.println("overrideHostNameVerifier - start");

    ((HttpsURLConnection) theURLConnection).setHostnameVerifier(new HostnameVerifier()
    {
        public boolean verify(String urlHost, String certHost)
        {
            if (!urlHost.equals(certHost))
            {
                System.out.println(
                    "certificate <" +
                    certHost +
                    "> does not match host <" +
                    urlHost +
                    "> but " +
                    "continuing anyway");
            }
            return true;
        }
    });
}
/**
 * Post MultiPart mime data to a web server.
 *
 * @exception MalformedURLException - thrown if URL invalid.
 * @exception IOException - thrown for any IO handling errors.
 */
public void postToWeb() throws MalformedURLException, IOException
{
    // Create URL to post to.
    URL destUrl = new URL(postUrl);

    // Prepare the connection for receiving the form
    URLConnection theURLConnection = destUrl.openConnection();

    //Setup HTTPS IP Tunneling through Proxy if needed.
    if (thereIsProxy)
    {
        setProxy(theURLConnection);
    }

    // Set URL options

```

```
theURLConnection.setDoOutput(true);
theURLConnection.setDoInput(true);
theURLConnection.setUseCaches(false);

// Override Certificate error for HTTPS
if (httpSwitch)
{
    // continue
}
else
{
    overrideHostNameVerifier(theURLConnection);
}

// Multi-Part delimiter (may want to randomly generate this)
String boundary = "7d021a37605f0";

// Set the Multi-Part Header
theURLConnection.setRequestProperty(
    "Content-Type",
    "multipart/form-data; boundary=" + boundary);

System.out.println("Before DataOutputStream");

DataOutputStream postFormFile =
    new DataOutputStream(theURLConnection.getOutputStream());

// Buffer used to build stream of multi-part form field
StringBuffer streamSend = new StringBuffer();

// Start Upload Parameters boundary section
streamSend.append("--" + boundary + "\r\n");

// Each Parameter is sent within a boundary delimiter
buildUploadParam(streamSend, "message", createXmlMessage(), boundary);

// End form parameters boundary section

// Begin File Upload section

buildUploadFileHeader(streamSend);

// Send Parameters and File header to web server.
postFormFile.write(streamSend.toString().getBytes());

System.out.println("Data Stream Sent\n" + streamSend.toString());
System.out.println("Before data Send");

// Write file to web server
try
{
    writeFileToWeb(postFormFile);
}
catch (IOException e)
{
    System.out.println("Exception in processing input file: " + fileName);
    System.out.println(e);
}

System.out.println("After file Sent");

// Terminate file multipart form-data POST boundary's
streamSend = new StringBuffer();
streamSend.append("\n--" + boundary + "\r\n");
postFormFile.write(streamSend.toString().getBytes());

System.out.println("Data Stream End Sent\n" + streamSend.toString());

// close/cleanup the https output stream
postFormFile.flush();
postFormFile.close();

// Process the web server response.
try
```

```

    {
        processWebResponse(theURLConnection);
    }
    catch (IOException e)
    {
        System.out.println("Exception in processing web server response ");
        System.out.println(e);
    }

    System.out.println("HttpMultiPartPost Done.");
}

/**
 * Get the input stream from the Web Server and
 * read the response. Write Response to standard out.
 *
 * @param theURLConnection URLConnection
 * @exception IOException - thrown for any IO handling errors.
 */
public void processWebResponse(URLConnection theURLConnection)
    throws IOException
{
    // Parse results to ensure file was sent ok."
    // Expecting: "Some sort of html or XML response/confirmation"
    // Read response from post
    // Initialize the input stream to be read from
    InputStream responseFromDestUrl = theURLConnection.getInputStream();

    // Build up response into a buffer
    StringBuffer thisResponsePage = new StringBuffer();
    byte[] respBuffer = new byte[4096];

    // The number of bytes read
    int bytesRead = 0;

    // Iterate to build up response
    while ((bytesRead = responseFromDestUrl.read(respBuffer)) >= 0)
    {
        thisResponsePage.append(new String(respBuffer).trim());
    }

    responseFromDestUrl.close(); // Close response stream

    // Display the response
    System.out.println("Web Server Response: Start" );
    System.out.println(thisResponsePage.toString());
    System.out.println("Web Server Response: End" );
}
/**
 * Prompt User for input parameters.
 * This could be done with a Java properties file or
 * an XML config file instead.
 */
public void promptInput()
{
    String currentLine;

    boolean processOptions = true;

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    try
    {
        System.out.print("User ID:  ");
        System.out.flush();
        currentLine = bufferedReader.readLine();
        if (currentLine != null && !currentLine.equals(""))
        {
            userId = currentLine;
        }

        System.out.println("Password: ");
        System.out.flush();
        currentLine = bufferedReader.readLine();
        if (currentLine != null && !currentLine.equals(""))
    }
}

```

```

{
    password = currentLine;
}

System.out.print("File Path and Name: ");
System.out.flush();
currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    fileName = currentLine;
    try
    {
        setSize();
    }
    catch (Exception e)
    {
        System.out.println(e);
        System.out.println("aborted");
    }
}

System.out.print("Proxy Server: (leave blank if none) ");
System.out.flush();
currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    proxyServer = currentLine;
    thereIsProxy = true;
}

if (thereIsProxy)
{
    System.out.print("Proxy User Id: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        proxyUser = currentLine;
    }

    System.out.print("Proxy Password: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        proxyPassword = currentLine;
    }
}
catch (IOException e)
{
    System.out.println("IOException " + e);
    System.exit(-1);
}
}

/**
 * Run the process to upload the file. If params are
 * passed from the command then they are processed by
 * acceptArgs, otherwise the user is prompted for input
 * from the promptInput method. Then the postToWeb
 * process is executed.
 *
 * @param args java.lang.String[]
 */
public void runProcess(String[] args)
{
    if (args.length > 0)
    {
        try
        {
            acceptArgs(args);
        }
        catch (Exception e)
        {

```

```

        System.out.println(e.getMessage());
        return;
    }
}
else
{
    promptInput();
}

try
{
    postToWeb();
}
catch (Exception e)
{
    System.out.println("Error uploading data, exception is: " + e);
    e.printStackTrace();
}

}

/**
 * Sets the file size of the upload file.
 *
 * @exception Exception - thrown if the input XML doesn't exist.
 */
public void setFileSize()
    throws Exception
{
    File checkFile = new File(fileName);

    if (checkFile.exists())
    {
        fileSize = "" + (checkFile.length());
    }
    else
    {
        throw new Exception(fileName + " File does not exist.");
    }
}
/**
 * Sets the HTTP switch to true if the request that is to be processed
 * in non SSL.  SSL requests is the default.
 *
 */
public void setHTTP()
{
    if (System.getProperty("http") == null)
    {
        httpSwitch = false;
    }
    else
    {
        if (System.getProperty("http").toString().equalsIgnoreCase("true"))
        {
            httpSwitch = true;
        }
        else
        {
            httpSwitch = false;
        }
    }
}
/**
 * Sets up the Proxy server properties for the URL Connection.
 * This assumes the proxy server requires authentication. Not
 * all Proxy servers do, so this will need to be changed. There
 * is also something called "Digest" which I believe is a different
 * kind of authentication some Proxy servers use that would have to
 * be handled as well. Port should be a parameter also.
 *
 * @param URLConnection urlConnection
 */

```

```
public void setProxy(URLConnection theURLConnection)
{
    if (theURLConnection instanceof com.sun.net.ssl.HttpsURLConnection)
    {
        ((com.sun.net.ssl.HttpsURLConnection) theURLConnection).setSSLSocketFactory(
            new SSLTunnelSocketFactory(proxyServer, proxyPort, proxyUser, proxyPassword));
    }
}
/***
 * Reads the file and write's it to the web server.
 *
 * @param postFormFile DataOutputStream
 * @exception IOException - thrown for any IO handling errors.
 */
public void writeFileToWeb(DataOutputStream postFormFile)
    throws java.io.IOException
{
    // Read/Write File as bytes

    FileInputStream uploadFileReader = new FileInputStream(fileName);
    int numBytesToRead = 1024;
    int availableBytesToRead;

    availableBytesToRead = uploadFileReader.available();

    System.out.println("size of file: " + availableBytesToRead);

    while ((availableBytesToRead = uploadFileReader.available()) > 0)
    {
        byte[] bufferBytesRead;
        // Adjust size of buffered bytes if necessary
        if (availableBytesToRead >= numBytesToRead)
        {
            bufferBytesRead = new byte[numBytesToRead];
        }
        else
        {
            bufferBytesRead = new byte[availableBytesToRead];
        }

        // Trap end of file condition
        int numberofBytesRead = uploadFileReader.read(bufferBytesRead);

        // Did we reach end of file
        if (numberofBytesRead == -1)
        {
            break;
        }

        // Write current buffered bytes to servlet
        postFormFile.write(bufferBytesRead);

    } // Iterate through contents of file
}
```

## Appendix B – Directory Listing sample program

### Directory Listing program example

```

package com.eds.ne.regional.samples;

import com.sun.net.ssl.HttpsURLConnection;
import com.sun.net.ssl.HostnameVerifier;
import java.net.*;
import java.io.*;

/**
 * This program is provided as sample only as is without warranty or
 * condition of any kind, either expressed or implied, including, but
 * not limited to, the implied warranties of merchantability and
 * fitness for a particular purpose.
 *
 * This sample Directory List program connects to the web server then
 * sends it an XML file containing the directory request message. The
 * name of the file with the XML message is passed to the program,
 * along with the server name. See the acceptArguments method for
 * parameters to pass on the command line. If no parameters are passed
 * the program will prompt the user for the input parameters then
 * execute. The directory listing response is simply displayed in
 * standard out.
 *
 * See the vendor specifications for details of the format of the XML message
 * for a directory listing. Below is a sample of the text that could
 * be in a file passed to this program to request a directory list.
 *
 *
* <?xml version="1.0" encoding="UTF-8"?>
* <DirectoryRequest>
*   <IdentificationHeader>
*     <TradingPartnerId>22222222</TradingPartnerId>
*     <UserId>xxx123</UserId>
*     <Password>dfdf534</Password>
*   </IdentificationHeader>
*   <Transaction>
*     <Function>DIRLIST</Function>
*     <FilesToReturnCount>50</FilesToReturnCount>
*     <SelectedFileTypes>
*       <FileType>997</FileType>
*       <FileType>835</FileType>
*     </SelectedFileTypes>
*     <FilesCreatedFromDate>20020102</FilesCreatedFromDate>
*     <FilesCreatedToDate>20020103</FilesCreatedToDate>
*   </Transaction>
* </DirectoryRequest>
*
*
* <P>
* In order to negotiate the SSL certificate of the web server the client has
* to have a trusted certificate in its Java keystore file. The web sites use
* GeoTrust Certificates. If this Certificate is not in your Java installations
* keystore you will need to install it. To install the certificate you can use
* the java keytool command
*
* This sample program was written using JDK 1.2.2. In order for the sample program
* to communicate over SSL the Java Secure Socket Extension (JSSE) needs to be installed.
* See http://java.sun.com/products/jsse/INSTALL.html.
* After installing JSSE add the following jar files must be in the class path before
* running the sample program: drive:/JSSE/jsse.jar;drive:/JSSE/jnet.jar;drive:/JSSE/jcert.jar
*   <P>
* <HR>
* <P>
* <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0" WIDTH="100%">
* <TR BGCOLOR="#9393ff">
*   <TD COLSPAN=4><FONT SIZE="+2"><B>Modification History</B></FONT></TD>
* </TR>
* <TR BGCOLOR="white" CLASS="TableRowColor">
*   <TH>Reason</TH><TH>Date</TH><TH>Systems Engineer</TH><TH>Description of Modification</TH>
* </TR>
* <TR>
```

```

*   <TD ALIGN= CENTER>NE Regional HIPAA Translator Implementation</TD>
*   <TD ALIGN= CENTER>12/19/2002</TD>
*   <TD ALIGN= CENTER>EDS</TD>
*   <TD>Initial deployment of this class.</TD>
*   </TR>
*   </TABLE>
*
*   @author Michael Smith
*
*   @see SSLTunnelSocketFactory
*/
public class ClientDirectoryList
{
/**
 * Authorized User Id. This User Id is authenticated
 * on the server before uploading the users message.
 */
    String userId = "";
/** 
 * User Password.
 */
    String password = "";
/** 
 * File drive:\path\name.
 */
    String fileName = "d:\\temp\\test.xml";
/** 
 * File size.
 */
    String fileSize = "0";
/** 
 * Users proxy servers name or ip address.
 */
    String proxyServer = null;
/** 
 * Users proxys port assignment.
 */
    String proxyPort = "80";
/** 
 * True if users sets their proxy server.
 */
    boolean thereIsProxy = false;
/** 
 * Users proxy server User Id.
 */
    String proxyUser = null;
/** 
 * Users proxy server password.
 */
    String proxyPassword = null;
/** 
 * HTTP Protocol Switch used to turn on HTTP only uploads.
 */
    boolean httpSwitch = false;
/** 
 * URL to post the data to. Defaults to test Web Server.
 */
    String postUrl = "https://almedicalprogram.alabama-medicaid.com/mod/secure
/WebDirectoryDownloadFromClient";
/**
 * ClientDirectoryList default constructor calls it's super, then
 * setups System properties for SSL protocol.
 */
public ClientDirectoryList()
{
    super();

    System.setProperty(
        "java.protocol.handler.pkgs",
        "com.sun.net.ssl.internal.www.protocol");

    // Required for ssl support and avoiding error:
    // "unknown protocol: https" exception
    java.security.Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
}

```

```

/**
 * Arguments passed on the command line are validated and
 * set by this method. The parameters are as follows:
 * <pre>
 * UserID Password DirectoryListXMLFile [URL] [proxyServer ProxyUser ProxyPassword]
 * </pre>
 * Parameters in brackets [] are optional.
 *
 * @exception Exception - Invalid arguments message.
 */
public void acceptArgs(String[] args)
    throws Exception
{
    if (args.length >= 3)
    {
        userId          = args[0];
        password        = args[1];
        fileName        = args[2];
    }

    setSize();

    if (args.length == 4)
    {
        postUrl        = args[3];
    }

    if (args.length == 6)
    {
        proxyServer     = args[3];
        proxyUser       = args[4];
        proxyPassword   = args[5];
        thereIsProxy   = true;
    }

    if (args.length == 7)
    {
        postUrl        = args[3];
        proxyServer     = args[4];
        proxyUser       = args[5];
        proxyPassword   = args[6];
        thereIsProxy   = true;
    }

    if (args.length == 3 || args.length == 4 || args.length == 6 || args.length == 7)
    {
        // continue
    }
    else
    {
        throw new Exception("Invalid number of parameters found:" + args.length + " must
be 3, 4, 6, or 7 \n" +
                           "Valid Params are: UserID Password FileName [URL] [proxyServer
ProxyUser ProxyPassword]");
    }
}

/**
 * Main creates an instance of ClientDirectoryList then
 * calls it's runProcess method.
 *
 * @param args java.lang.String[]
 */
public static void main(String[] args)
{
    ClientDirectoryList clientDirectoryList = new ClientDirectoryList();
    clientDirectoryList.setHTTP();
    clientDirectoryList.runProcess(args);
}

/**
 * Override Hostname Verifier to ignore Certificate errors where the

```

```

* URL on the certificate doesn't match the URL being accessed.
*
* @param theURLConnection URLConnection
*/
public void overrideHostNameVerifier(URLConnection theURLConnection)
{
    ((HttpsURLConnection) theURLConnection).setHostnameVerifier(new HostnameVerifier()
    {
        public boolean verify(String urlHost, String certHost)
        {
            if (!urlHost.equals(certHost))
            {
                System.out.println(
                    "certificate <" +
                    certHost +
                    "> does not match host <" +
                    urlHost +
                    "> but " +
                    "continuing anyway");
            }
            return true;
        }
    });
}
/** 
 * Post to a web server.
*
* @exception MalformedURLException - thrown if URL invalid.
* @exception IOException - thrown for any IO handling errors.
*/
public void postToWeb() throws MalformedURLException, IOException
{
    // Create URL to post to.
    URL destUrl = new URL(postUrl);

    // Prepare the connection for receiving the form
    URLConnection theURLConnection = destUrl.openConnection();

    //Setup HTTPS IP Tunneling through Proxy if needed.
    if (thereIsProxy)
    {
        setProxy(theURLConnection);
    }

    // Set URL options
    theURLConnection.setDoOutput(true);
    theURLConnection.setDoInput(true);
    theURLConnection.setUseCaches(false);

    // Setup Override of error name error if using HTTPS
    // and the certificate name doesn't match the URL.
    // This code should be removed if the certificate name matches.
    if (httpSwitch)
    {
        // continue
    }
    else
    {
        // Override Certificate error
        overrideHostNameVerifier(theURLConnection);
    }

    // Set the XML Post Header
    theURLConnection.setRequestProperty(
        "Content-Type",
        "text/xml");

    System.out.println("Before DataOutputStream");

    DataOutputStream postFormFile =
        new DataOutputStream(theURLConnection.getOutputStream());

    System.out.println("Before data Send");

    // Write XML file to web server
}

```

```

try
{
    writeXmlRequestToWeb(postFormFile);
}
catch (IOException e)
{
    System.out.println("Exception in processing input file: " + fileName);
    System.out.println(e);
}

System.out.println("After file Sent");

// close/cleanup the https output stream
postFormFile.flush();
postFormFile.close();

// Process the web server response.
try
{
    processWebResponse(theURLConnection);
}
catch (IOException e)
{
    System.out.println("Exception in processing web server response ");
    System.out.println(e);
}

System.out.println("HttpMultiPartPost Done.");

}
/***
 * Get the input stream from the Web Server and
 * read the response. Write Response to standard out.
 *
 * @param theURLConnection URLConnection
 * @exception IOException - thrown for any IO handling errors.
 */
public void processWebResponse(URLConnection theURLConnection)
    throws IOException
{
    // Parse results to ensure file was sent ok."
    // Expecting: "Some sort of html or XML response/confirmation"
    // Read response from post
    // Initialize the input stream to be read from
    InputStream responseFromDestUrl = theURLConnection.getInputStream();

    // Build up response into a buffer
    StringBuffer thisResponsePage = new StringBuffer();
    byte[] respBuffer = new byte[4096];

    // The number of bytes read
    int bytesRead = 0;

    // Iterate to build up response
    while ((bytesRead = responseFromDestUrl.read(respBuffer)) >= 0)
    {
        thisResponsePage.append(new String(respBuffer).trim());
    }

    responseFromDestUrl.close(); // Close response stream

    // Display the response
    System.out.println("Web Server Response: Start" );
    System.out.println(thisResponsePage.toString());
    System.out.println("Web Server Response: End" );
}
/***
 * Prompt User for input parameters.
 * This could be done with a Java properties file or
 * an XML config file instead.
 */
public void promptInput()
{
    String currentLine;

```

```

boolean processOptions = true;

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
try
{
    System.out.print("User ID: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        userId = currentLine;
    }

    System.out.println("Password: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        password = currentLine;
    }

    System.out.print("XML File Path and Name: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        fileName = currentLine;
        try
        {
            setFileSize();
        }
        catch (Exception e)
        {
            System.out.println(e);
            System.out.println("aborted");
        }
    }

    System.out.print("Proxy Server: (leave blank if none) ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        proxyServer = currentLine;
        thereIsProxy = true;
    }

    if (thereIsProxy)
    {
        System.out.print("Proxy User Id: ");
        System.out.flush();
        currentLine = bufferedReader.readLine();
        if (currentLine != null && !currentLine.equals(""))
        {
            proxyUser = currentLine;
        }

        System.out.print("Proxy Password: ");
        System.out.flush();
        currentLine = bufferedReader.readLine();
        if (currentLine != null && !currentLine.equals(""))
        {
            proxyPassword = currentLine;
        }
    }
}
catch (IOException e)
{
    System.out.println("IOException " + e);
    System.exit(-1);
}
}

/**
 * Run the process to upload the file. If params are
 * passed from the command then they are processed by

```

```

/*
 * acceptArgs otherwise the user is prompted for input
 * from the promptInput method. Then the postToWeb
 * process is executed.
 *
 * @param args java.lang.String[]
 */
public void runProcess(String[] args)
{
    if (args.length > 0)
    {
        try
        {
            acceptArgs(args);
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            return;
        }
    }
    else
    {
        promptInput();
    }

    try
    {
        postToWeb();
    }
    catch (Exception e)
    {
        System.out.println("Error uploading data, exception is: " + e);
        e.printStackTrace();
    }
}

/**
 * Sets the file size of the upload file.
 *
 * @exception Exception - thrown if the input XML doesn't exist.
 */
public void setFileSize()
    throws Exception
{
    File checkFile = new File(fileName);

    if (checkFile.exists())
    {
        fileSize = "" + (checkFile.length());
    }
    else
    {
        throw new Exception(fileName + " File does not exist.");
    }
}
/**
 * Sets the HTTP switch to true if the request that is to be processed
 * is not SSL. SSL requests is the default.
 */
public void setHTTP()
{
    if (System.getProperty("http") == null)
    {
        httpSwitch = false;
    }
    else
    {
        if (System.getProperty("http").toString().equalsIgnoreCase("true"))
        {
            httpSwitch = true;
        }
    }
}

```

```

        else
        {
            httpSwitch = false;
        }
    }
}**
 * Sets up the Proxy server properties for the URL Connection.
 * This assumes the proxy server requires authentication. Not
 * all Proxy servers do, so this will need to be changed. There
 * is also something called "Digest" which I believe is a different
 * kind of authentication some Proxy servers use that would have to
 * be handled as well. The Port number should be a parameter also.
 *
 * @param theURLConnection URLConnection
 */
public void setProxy(URLConnection theURLConnection)
{
    if (theURLConnection instanceof com.sun.net.ssl.HttpsURLConnection)
    {
        ((com.sun.net.ssl.HttpsURLConnection) theURLConnection).setSSLSocketFactory(
            new SSLTunnelSocketFactory(proxyServer, proxyPort, proxyUser, proxyPassword));
    }
}
/***
 * Writes the XML Request for a directory list to web server.
 *
 * @param postFormFile DataOutputStream
 * @exception IOException - thrown for any IO handling errors.
 */
public void writeXmlRequestToWeb(DataOutputStream postFormFile)
    throws java.io.IOException
{
    // Read/Write File as bytes

    FileInputStream uploadFileReader = new FileInputStream(fileName);
    int numBytesToRead = 1024;
    int availableBytesToRead;

    availableBytesToRead = uploadFileReader.available();

    System.out.println("size of file: " + availableBytesToRead);

    while ((availableBytesToRead = uploadFileReader.available()) > 0)
    {
        byte[] bufferBytesRead;
        // Adjust size of buffered bytes if necessary
        if (availableBytesToRead >= numBytesToRead)
        {
            bufferBytesRead = new byte[numBytesToRead];
        }
        else
        {
            bufferBytesRead = new byte[availableBytesToRead];
        }

        // Trap end of file condition
        int numberOfBytesRead = uploadFileReader.read(bufferBytesRead);

        // Did we reach end of file
        if (numberOfBytesRead == -1)
        {
            break;
        }

        // Write current buffered bytes to servlet
        postFormFile.write(bufferBytesRead);

    } // Iterate through contents of file
}
}

```

## Appendix C – Download a File sample program

### Download a file program example

```

package com.eds.ne.regional.samples;

import com.sun.net.ssl.HttpsURLConnection;
import com.sun.net.ssl.HostnameVerifier;
import java.net.*;
import java.io.*;

/**
 * This program is provided as sample only as is without warranty or
 * condition of any kind, either expressed or implied, including, but
 * not limited to, the implied warranties of merchantability and
 * fitness for a particular purpose.
 *
 * This sample File Download program connects to the web server then
 * sends it an XML file containing the download request message. The
 * name of the file with the XML message is passed to the program,
 * along with the server name. See the acceptArguments method for
 * parameters to pass on the command line. If no parameters are passed
 * the program will prompt the user for the input parameters then
 * execute. The downloaded file is simply displayed in standard out.
 *
 * See the vendor specifications for details of the format of the XML message
 * for a file download. Below is a sample of the text that could
 * be in a file passed to this program to request a directory list.
 *
 *
 * <?xml version="1.0" encoding="UTF-8" ?>
* <DownloadRequest>
*   <IdentificationHeader>
*     <TradingPartnerId>22222222</TradingPartnerId>
*     <UserId>xyzuser</UserId>
*     <Password>jmstp567</Password>
*   </IdentificationHeader>
*   <Transaction>
*     <Function>DOWNLOAD</Function>
*     <FileName>000000123</FileName>
*     <FileFormat>TXT</FileFormat>
*   </Transaction>
* </DownloadRequest>
*
*
* <P>
* In order to negotiate the SSL certificate of the web server the client has
* to have a trusted certificate in its Java keystore file. The web sites use
* GeoTrust Certificates. If this Certificate is not in your Java installations
* keystore you will need to install it. To install the certificate you can use
* the java keytool command.
*
* This sample program was written using JDK 1.2.2. In order for the sample program
* to communicate over SSL the Java Secure Socket Extension (JSSE) needs to be installed.
* See http://java.sun.com/products/jsse/INSTALL.html.
* After installing JSSE add the following jar files must be in the class path before
* running the sample program: drive:/JSSE/jsse.jar;drive:/JSSE/jnet.jar;drive:/JSSE/jcert.jar
*   <P>
* <HR>
* <P>
* <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0" WIDTH="100%">
* <TR BGCOLOR="#9393ff">
*   <TD COLSPAN=4><FONT SIZE="+2"><B>Modification History</B></FONT></TD>
* </TR>
* <TR BGCOLOR="white" CLASS="TableRowColor">
*   <TH>Reason</TH><TH>Date</TH><TH>Systems Engineer</TH><TH>Description of Modification</TH>
* </TR>
* <TR>
*   <TD ALIGN=LEFT>NE Regional HIPAA Translator Implementation</TD>
*   <TD ALIGN=LEFT>12/19/2002</TD>
*   <TD ALIGN=LEFT>EDS</TD>
*   <TD>Initial deployment of this class.</TD>
* </TR>
* </TABLE>
```

```

/*
 *   @author Michael Smith
 *
 *   @see SSLTunnelSocketFactory
 */
public class ClientFileDownload
{
/**
 * Users Id. This User Id is authenticated
 * on the server before uploading the users message.
 */
    String userId = "";

/**
 * User password.
 */
    String password = "";

/**
 * Download file name.
 */
    String fileName = "";

/**
 * Message file size.
 */
    String fileSize = "0";

/**
 * Users proxy servers name or ip address.
 */
    String proxyServer = null;

/**
 * Users proxys port assignment.
 */
    String proxyPort = "80";

/**
 * True if Users sets their proxy server.
 */
    boolean thereIsProxy = false;

/**
 * Users proxy server User Id.
 */
    String proxyUser = null;

/**
 * Users Proxy Server Password.
 */
    String proxyPassword = null;

/**
 * HTTP protocal switch used to turn on HTTP only uploads.
 */
    boolean httpSwitch = false;

/**
 * URL to post the data to. Defaults to test Web Server
 */
    String postUrl = "https://almedicalprogram.alabama-medicaid.com/mod/secure
/WebDirectoryDownloadFromClient";
/**
 * ClientFileDownload default constructor calls it's super, then
 * setups System properties for SSL protocol.
 */
public ClientFileDownload()
{
    super();

    System.setProperty(
        "java.protocol.handler.pkgs",
        "com.sun.net.ssl.internal.www.protocol");

    // Required for ssl support and avoiding error:
    // "unknown protocol: https" exception
    java.security.Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
}

/**
 * Arguments passed on the command line are validated and
 * set by this method. The parameters are as follows:
 * <pre>
 * UserID Password FileName [URL] [proxyServer ProxyUser ProxyPassword]
 * </pre>

```

```

/*
 * @param args java.lang.String[]
 * @exception Exception - Invalid arguments message.
 */
public void acceptArgs(String[] args)
    throws Exception
{
    if (args.length >= 3)
    {
        userId      = args[0];
        password    = args[1];
        fileName    = args[2];
    }

    setFileSize();

    if (args.length == 4)
    {
        postUrl     = args[3];
    }

    if (args.length == 6)
    {
        proxyServer   = args[3];
        proxyUser     = args[4];
        proxyPassword = args[5];
        thereIsProxy = true;
    }

    if (args.length == 7)
    {
        postUrl      = args[3];
        proxyServer   = args[4];
        proxyUser     = args[5];
        proxyPassword = args[6];
        thereIsProxy = true;
    }

    if (args.length == 3 || args.length == 4 || args.length == 6 || args.length == 7)
    {
        // continue
    }
    else
    {
        throw new Exception("Invalid number of parameters found:" + args.length + " must
be 3, 4, 6, or 7 \n" +
                           "Valid Params are: UserID Password FileName [URL] [proxyServer
ProxyUser ProxyPassword]");
    }
}

/**
 * Main creates an instance of ClientFileDownload then
 * calls it's runProcess method.
 *
 * @param args java.lang.String[]
 */
public static void main(String[] args)
{
    ClientFileDownload clientFileDownload = new ClientFileDownload();
    clientFileDownload.setHTTP();
    clientFileDownload.runProcess(args);
}

/**
 * Override Hostname Verifier to ignore Certificate errors where the
 * URL on the certificate doesn't match the URL being accessed.
 *
 * @param theURLConnection URLConnection
 */
public void overrideHostNameVerifier(URLConnection theURLConnection)
{
}

```

```

((HttpsURLConnection) theURLConnection).setHostnameVerifier(new HostnameVerifier()
{
    public boolean verify(String urlHost, String certHost)
    {
        if (!urlHost.equals(certHost))
        {
            System.out.println(
                "certificate <" +
                certHost +
                "> does not match host <" +
                urlHost +
                "> but " +
                "continuing anyway");
        }
        return true;
    }
});

/***
 * Post a request for a file download.
 *
 * @exception MalformedURLException - thrown if URL invalid.
 * @exception IOException - thrown for any IO handling errors.
 */
public void postToWeb() throws MalformedURLException, IOException
{
    // Create URL to post to.
    URL destUrl = new URL(postUrl);

    // Prepare the connection for receiving the form
    URLConnection theURLConnection = destUrl.openConnection();

    //Setup HTTPS IP Tunneling through Proxy if needed.
    if (thereIsProxy)
    {
        setProxy(theURLConnection);
    }

    // Set URL options
    theURLConnection.setDoOutput(true);
    theURLConnection.setDoInput(true);
    theURLConnection.setUseCaches(false);

    if (httpSwitch)
    {
        // continue
    }
    else
    {
        // Override Certificate error
        overrideHostNameVerifier(theURLConnection);
    }

    // Set the XML Post Header
    theURLConnection.setRequestProperty(
        "Content-Type",
        "text/xml");

    System.out.println("Before DataOutputStream");

    DataOutputStream postFormFile =
        new DataOutputStream(theURLConnection.getOutputStream());

    System.out.println("Before data Send");

    // Write file to web server
    try
    {
        writeXmlRequestToWeb(postFormFile);
    }
    catch (IOException e)
    {
        System.out.println("Exception in processing input file: " + fileName);
        System.out.println(e);
    }
}

```

```

}

System.out.println("After file Sent");

// close/cleanup the https output stream
postFormFile.flush();
postFormFile.close();

// Process the web server response.
try
{
    processWebResponse(theURLConnection);
}
catch (IOException e)
{
    System.out.println("Exception in processing web server response ");
    System.out.println(e);
}

System.out.println("postToWeb Done.");

}
/***
 * Get the input stream from the Web Server and
 * read the response. Write Response to standard out.
 *
 * @param theURLConnection URLConnection
 * @exception IOException - thrown for any IO handling errors.
 */
public void processWebResponse(URLConnection theURLConnection)
    throws IOException
{
    // Parse results to ensure file was sent ok."
    // Expecting: "Some sort of html or XML response/confirmation"
    // Read response from post
    // Initialize the input stream to be read from
    InputStream responseFromDestUrl = theURLConnection.getInputStream();

    // Build up response into a buffer
    StringBuffer thisResponsePage = new StringBuffer();
    byte[] respBuffer = new byte[4096];

    // The number of bytes read
    int bytesRead = 0;

    // Iterate to build up response
    while ((bytesRead = responseFromDestUrl.read(respBuffer)) >= 0)
    {
        thisResponsePage.append(new String(respBuffer).trim());
    }

    responseFromDestUrl.close(); // Close response stream

    // Display the response
    System.out.println("Web Server Response: Start" );
    System.out.println(thisResponsePage.toString());
    System.out.println("Web Server Response: End" );
}
/***
 * Prompt User for input parameters.
 * This could be done with a Java properties file or
 * an XML config file instead.
 */
public void promptInput()
{
    String currentLine;

    boolean processOptions = true;

    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    try
    {
        System.out.print("User ID:  ");
        System.out.flush();
    }

```

```

currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    userId = currentLine;
}

System.out.println("Password: ");
System.out.flush();
currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    password = currentLine;
}

System.out.print("XML File Path and Name:  ");
System.out.flush();
currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    fileName = currentLine;
    try
    {
        setSize();
    }
    catch (Exception e)
    {
        System.out.println(e);
        System.out.println("aborted");
    }
}

System.out.print("Proxy Server: (leave blank if none)  ");
System.out.flush();
currentLine = bufferedReader.readLine();
if (currentLine != null && !currentLine.equals(""))
{
    proxyServer = currentLine;
    thereIsProxy = true;
}

if (thereIsProxy)
{
    System.out.print("Proxy User Id: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        proxyUser = currentLine;
    }

    System.out.print("Proxy Password: ");
    System.out.flush();
    currentLine = bufferedReader.readLine();
    if (currentLine != null && !currentLine.equals(""))
    {
        proxyPassword = currentLine;
    }
}
catch (IOException e)
{
    System.out.println("IOException " + e);
    System.exit(-1);
}
}

/**
 * Run the process to upload the file.  If params are
 * passed from the command then they are processed by
 * acceptArgs otherwise the user is prompted for input
 * from the promptInput method.  Then the postToWeb
 * process is executed.
 *
 * @param args java.lang.String[]
 */

```

```

public void runProcess(String[] args)
{
    if (args.length > 0)
    {
        try
        {
            acceptArgs(args);
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            return;
        }
    }
    else
    {
        promptInput();
    }

    try
    {
        postToWeb();
    }
    catch (Exception e)
    {
        System.out.println("Error uploading data, exception is: " + e);
        e.printStackTrace();
    }
}

/**
 * Sets the file size of the upload file.
 *
 * @exception Exception - thrown if the input XML doesn't exist.
 */
public void setFileSize()
    throws Exception
{
    File checkFile = new File(fileName);

    if (checkFile.exists())
    {
        fileSize = "" + (checkFile.length());
    }
    else
    {
        throw new Exception(fileName + " File does not exist.");
    }
}
/**
 * Sets the HTTP switch to true if the request that is to be processed
 * in non SSL.  SSL requests is the default.
 *
 */
public void setHTTP()
{
    if (System.getProperty("http") == null)
    {
        httpSwitch = false;
    }
    else
    {
        if (System.getProperty("http").toString().equalsIgnoreCase("true"))
        {
            httpSwitch = true;
        }
        else
        {
            httpSwitch = false;
        }
    }
}

```

```

/**
 * Sets up the Proxy server properties for the URL Connection.
 * This assumes the proxy server requires authentication. Not
 * all Proxy servers do, so this will need to be changed. There
 * is also something called "Digest" which I believe is a different
 * kind of authentication some Proxy servers use that would have to
 * be handled as well. Port should be a parameter also.
 *
 * @param theURLConnection URLConnection
 */
public void setProxy(URLConnection theURLConnection)
{
    if (theURLConnection instanceof com.sun.net.ssl.HttpsURLConnection)
    {
        ((com.sun.net.ssl.HttpsURLConnection) theURLConnection).setSSLSocketFactory(
            new SSLTunnelSocketFactory(proxyServer, proxyPort, proxyUser, proxyPassword));
    }
}
/**
 * Writes the XML Request for a file to web server.
 *
 * @param postFormFile DataOutputStream
 * @exception java.io.IOException The exception description.
 */
public void writeXmlRequestToWeb(DataOutputStream postFormFile)
    throws java.io.IOException
{
    // Read/Write File as bytes

    FileInputStream uploadFileReader = new FileInputStream(fileName);
    int numBytesToRead = 1024;
    int availableBytesToRead;

    availableBytesToRead = uploadFileReader.available();

    System.out.println("size of file: " + availableBytesToRead);

    while ((availableBytesToRead = uploadFileReader.available()) > 0)
    {
        byte[] bufferBytesRead;
        // Adjust size of buffered bytes if necessary
        if (availableBytesToRead >= numBytesToRead)
        {
            bufferBytesRead = new byte[numBytesToRead];
        }
        else
        {
            bufferBytesRead = new byte[availableBytesToRead];
        }

        // Trap end of file condition
        int numberOfBytesRead = uploadFileReader.read(bufferBytesRead);

        // Did we reach end of file
        if (numberOfBytesRead == -1)
        {
            break;
        }

        // Write current buffered bytes to servlet
        postFormFile.write(bufferBytesRead);

    } // Iterate through contents of file
}
}

```

## Appendix D – SSL Tunnel through proxy server utility sample program

Sample program only needed if having to connect through a Proxy Server to get to the Internet.

```

package com.eds.ne.regional.samples;

import java.net.*;
import java.io.*;
import java.security.*;
import sun.misc.BASE64Encoder;
import javax.net.*;
import javax.net.ssl.*;
/**
 * This program is provided as sample only as is without warranty or
 * condition of any kind, either expressed or implied, including, but
 * not limited to, the implied warranties of merchantability and
 * fitness for a particular purpose.
 *
 * This sample program is used to facilitate HTTPS (SSL) communication when
 * connecting through a Proxy Server.
 *
 *      <P>
 *      <HR>
 *      <P>
 *      <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0" WIDTH="100%">
 *      <TR BGCOLOR="#9393ff">
 *          <TD COLSPAN=4><FONT SIZE="+2"><B>Modification History</B></FONT></TD>
 *      </TR>
 *      <TR BGCOLOR="white" CLASS="TableRowColor">
 *          <TH>Reason</TH><TH>Date</TH><TH>Systems Engineer</TH><TH>Description of Modification</TH>
 *      </TR>
 *      <TR>
 *          <TD ALIGN=CENTER>NE Regional HIPAA Translator Implementation</TD>
 *          <TD ALIGN=CENTER>12/19/2002</TD>
 *          <TD ALIGN=CENTER>EDS</TD>
 *          <TD>Initial deployment of this class.</TD>
 *      </TR>
 *  </TABLE>
 *
 *  @see SSLTunnelSocketFactory
 */
public class SSLTunnelSocketFactory extends SSLSocketFactory
{
    /**
     * Proxy server Host name.
     */
    private String tunnelHost;
    /**
     * Proxy server port assignment.
     */
    private int tunnelPort;
    /**
     * SSL Socket Factory returns a default Socket configured for SSL communication.
     */
    private SSLSocketFactory dfactory;
    /**
     * Proxy Server password used in tunneling.
     */
    private String tunnelPassword;
    /**
     * Proxy Server User Id used in tunneling.
     */
    private String tunnelUserName;
    /**
     * Switch to indicate we have a connection.
     */
    private boolean socketConnected = false;

    /**
     * Constructor for the SSLTunnelSocketFactory object.
     *
     * @param proxyHost The url of the proxy host
     * @param proxyPort the port of the proxy
     */
}

```

```

public SSLTunnelSocketFactory(String proxyHost, String proxyPort)
{
    System.out.println("SSLTunnelSocketFactory: Creating Socket Factory");

    tunnelHost = proxyHost;
    tunnelPort = Integer.parseInt(proxyPort);

    dfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
}
/**
 * Default constructor for the SSLTunnelSocketFactory object.
 *
 * @param proxyHost      The url of the proxy host
 * @param proxyPort      the port of the proxy
 * @param proxyUserName  username for authenticating with the proxy
 * @param proxyPassword  password for authenticating with the proxy
 */
public SSLTunnelSocketFactory(
    String proxyHost,
    String proxyPort,
    String proxyUserName,
    String proxyPassword)
{
    System.err.println("creating Socket Factory with password/username");
    tunnelHost = proxyHost;
    tunnelPort = Integer.parseInt(proxyPort);
    tunnelUserName = proxyUserName;
    tunnelPassword = proxyPassword;
    dfactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
}
/**
 * Creates a new SSL Tunneled Socket
 *
 * @param host           destination host
 * @param port           destination port
 * @return               tunneled SSL Socket
 * @exception IOException raised by IO error
 * @exception UnknownHostException raised when the host is unknown
 */
public Socket createSocket(String host, int port)
    throws IOException, UnknownHostException
{
    return createSocket(null, host, port, true);
}
/**
 * Creates a new SSL Tunneled Socket
 *
 * @param host           Destination Host
 * @param port           Destination Port
 * @param clientHost    Ignored
 * @param clientPort    Ignored
 * @return               SSL Tunneled Socket
 * @exception IOException Raised when IO error occurs
 * @exception UnknownHostException Raised when the destination host is unknown
 */
public Socket createSocket(
    String host,
    int port,
    InetAddress clientHost,
    int clientPort)
    throws IOException, UnknownHostException
{
    return createSocket(null, host, port, true);
}
/**
 * Creates a new SSL Tunneled Socket.
 *
 * @param host   destination host
 * @param port   destination port
 * @return     tunneled SSL Socket
 * @exception IOException raised when IO error occurs
 */
public Socket createSocket(InetAddress host, int port)
    throws IOException
{
}

```

```

        return createSocket(null, host.getHostName(), port, true);
    }
    /**
     * Creates a new SSL Tunneled Socket
     *
     * @param      address          destination host
     * @param      port             destination port
     * @param      clientAddress    ignored
     * @param      clientPort       ignored
     * @return     tunneled SSL Socket
     * @exception  IOException      raised when IO exception occurs
    */
    public Socket createSocket(
        InetAddress address,
        int port,
        InetAddress clientAddress,
        int clientPort)
        throws IOException
    {
        return createSocket(null, address.getHostName(), port, true);
    }
    /**
     * Creates a new SSL Tunneled Socket
     *
     * @param      s                Ignored
     * @param      host              destination host
     * @param      port              destination port
     * @param      autoClose         whether to close the socket automatically
     * @return     proxy tunneled socket
     * @exception  IOException      raised by an IO error
     * @exception  UnknownHostException  raised when the host is unknown
    */
    public Socket createSocket(Socket s, String host, int port, boolean autoClose)
        throws IOException, UnknownHostException
    {
        Socket tunnel = new Socket(tunnelHost, tunnelPort);
        doTunnelHandshake(tunnel, host, port);

        SSLSocket result =
            (SSLSocket) dfactory.createSocket(tunnel, host, port, autoClose);

        result.addHandshakeCompletedListener(new HandshakeCompletedListener()
        {
            public void handshakeCompleted(HandshakeCompletedEvent event)
            {
                System.out.println("Handshake Finished!");
                System.out.println("\t CipherSuite :" + event.getCipherSuite());
                System.out.println("\t SessionId: " + event.getSession());
                System.out.println("\t PeerHost: " + event.getSession().getPeerHost());
                setSocketConnected(true);
            }
        });
        // thanks to David Lord in the java forums for figuring out this line is the problem
        // result.startHandshake(); //this line is the bug which stops Tip111 from working correctly
        return result;
    }
    /**
     * Description of the Method
     *
     * @param      tunnel           tunnel socket
     * @param      host              destination host
     * @param      port              destination port
     * @exception  IOException      raised when an IO error occurs
    */
    private void doTunnelHandshake(Socket tunnel, String host, int port)
        throws IOException
    {
        OutputStream out = tunnel.getOutputStream();

        //generate connection string
        String msg =
            "CONNECT "
            + host

```

```

+ ":"+
+ port
+ " HTTP/1.0\n"
+ "User-Agent: "
+ sun.net.www.protocol.http.HttpURLConnection.userAgent;

if (tunnelUserName != null && tunnelPassword != null)
{
    //add basic authentication header for the proxy
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    String encodedPassword = enc.encode((tunnelUserName +
                                         ":" + tunnelPassword).getBytes());
    msg = msg + "\nProxy-Authorization: Basic " + encodedPassword;
}

msg = msg + "\nContent-Length: 0";
msg = msg + "\nPragma: no-cache";
msg = msg + "\r\n\r\n";
System.out.println("doTunnelHandshake Message:" + msg);

byte b[];
try
{
    //we really do want ASCII7 as the http protocol doesnt change with locale
    b = msg.getBytes("ASCII7");
}
catch (UnsupportedEncodingException ignored)
{
    //If ASCII7 isn't there, something is seriously wrong!
    b = msg.getBytes();
}

out.write(b);
out.flush();

byte reply[] = new byte[200];
int replyLen = 0;
int newlinesSeen = 0;
boolean headerDone = false;

InputStream in = tunnel.getInputStream();

boolean error = false;

while (newlinesSeen < 2)
{
    int i = in.read();
    if (i < 0)
    {
        throw new IOException("Unexpected EOF from Proxy");
    }

    if (i == '\n')
    {
        headerDone = true;
        ++newlinesSeen;
    }
    else
    {
        if (i != '\r')
        {
            newlinesSeen = 0;
            if (!headerDone && replyLen < reply.length)
            {
                reply[replyLen++] = (byte) i;
            }
        }
    }
}

//convert byte array to string
String replyStr;

try

```

```

{
    replyStr = new String(reply, 0, replyLen, "ASCII7");
}
catch (UnsupportedEncodingException ignored)
{
    replyStr = new String(reply, 0, replyLen);
}

//we check for connection established because our proxy returns http/1.1 instead of 1.0
if (replyStr.toLowerCase().indexOf("200 connection established") == -1)
{
    System.out.println("doTunnelHandshake:" + replyStr);
    throw new IOException(
        "Unable to tunnel through "
        + tunnelHost
        + ":"
        + tunnelPort
        + ". Proxy returns\""
        + replyStr
        + "\"");
}
//tunneling handshake was successful
}
/**
 * Gets the defaultCipherSuites attribute of the SSLTunnelSocketFactory
 * object.
 *
 * @return      The defaultCipherSuites value
 */
public String[] getDefaultCipherSuites()
{
    return dfactory.getDefaultCipherSuites();
}
/**
 * Gets the socketConnected attribute of the SSLTunnelSocketFactory object
 *
 * @return      The socketConnected value
 */
public synchronized boolean getSocketConnected()
{
    return socketConnected;
}
/**
 * Gets the supportedCipherSuites attribute of the SSLTunnelSocketFactory
 * object.
 *
 * @return      The supportedCipherSuites value
 */
public String[] getSupportedCipherSuites()
{
    return dfactory.getSupportedCipherSuites();
}
/**
 * Sets the proxyPassword attribute of the SSLTunnelSocketFactory object
 *
 * @param proxyPassword  The new proxyPassword value
 */
public void setProxyPassword(String proxyPassword)
{
    tunnelPassword = proxyPassword;
}
/**
 * Sets the proxyUserName attribute of the SSLTunnelSocketFactory object
 *
 * @param proxyUserName  The new proxyUserName value
 */
public void setProxyUserName(String proxyUserName)
{
    tunnelUserName = proxyUserName;
}
/**
 * Sets the socketConnected attribute of the SSLTunnelSocketFactory object
 *
 * @param b  The new socketConnected value
 */

```

```
private synchronized void setSocketConnected(boolean b)
{
    socketConnected = b;
}
```

## Appendix E – Setting up a RAS Connection

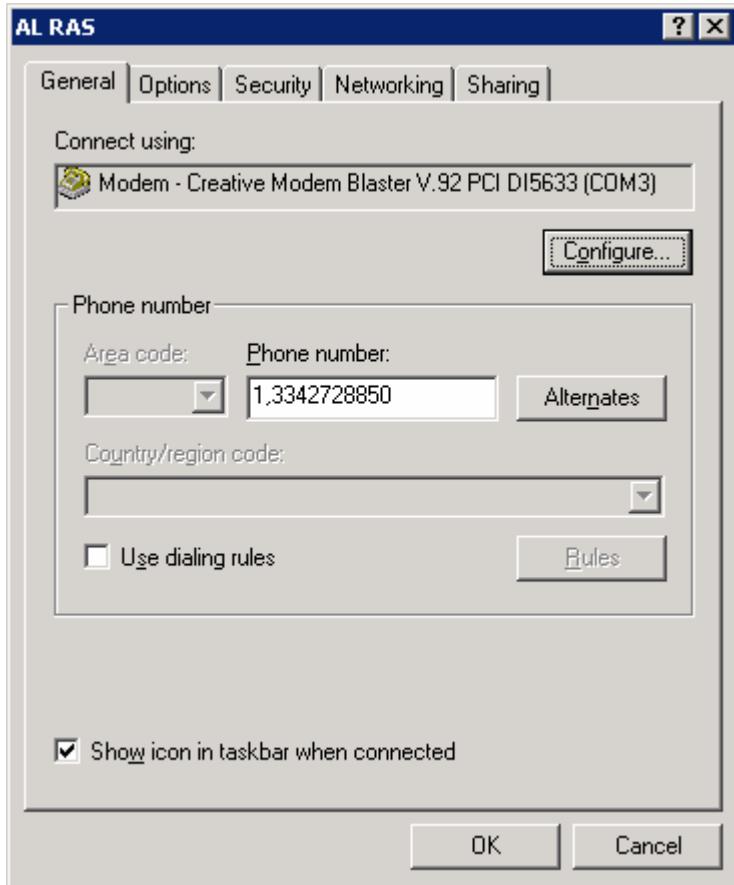
The following screen prints illustrate the settings specific to the Alabama Medicaid RAS. The User name and password are both Medicaid.



Now, click the properties button to obtain the window pictured on the next page.

If your phone line is located within the Montgomery, Alabama area, the 1 and area code will not be required with the phone number. Depending on your phone system and its access to a outside line or long distance service, other modifications to the Phone number could be required.

Configure your Modem according to the modem specifications.



Now, click on the Networking Tab to obtain the window on the next page.

Highlight the Internet Protocol (TCP/IP) entry, and then click the properties button. Enter the DNS server address 10.1.1.17 as shown.

